Sightseeing, venues, and friends: Predictive analytics with Spark ML and Cassandra

Natalino Busa
Head of Applied Data Science at Teradata
Location Based Transaction Networks

Locations

Network

Transactions
You find them everywhere!

Emails

Social Networks
Check-ins

Payments
You find them everywhere!

- Emails
- Industrial IoT networks
- Social Networks
  - Check-ins
- Social Networks
  - Ratings
- Payments
- Calls
You find them everywhere!

- Emails
- Social Networks
- Check-ins
- Payments
- Industrial IoT networks
- Social Networks
- Ratings
- Transportation
- Traveling Services
- Calls
- Messaging
- Remote Assistance

@natbusa | linkedin.com: Natalino Busa
Use cases

- Clustering
- Classification
- Anomaly Detection
- Fraud Detection
- Cyber Security
- Forecasting
- Process Mining
- Communities
Location Based Transaction Networks

Locations

Network

Transactions
Evolution of ETL

**Extract**
- Streaming Real-Time Events
- Batch Extracts from DBs
- Consume from APIs
- Extract from no-SQL datastores

**Transform**
- Aggregate
- Machine Learning
- Score, Train
- SQL, Graph, ML

**Load**
- Expose as APIs
- Load to DBs
- Produce to APIs
- Push to noSQL datastores
Kafka+Cassandra+Spark: Streaming Machine Learning

**Kafka**
- Streaming Events
- Distributed, Scalable Transport
- Events are persisted
- Decoupled Consumer-Producers
- Topics and Partitions

**Cassandra**
- Fast writes
- 2D Data Structure
- Replicated
- Tunable consistency
- Multi-Data centers

**Spark**
- Ad-Hoc Queries
- Joins, Aggregate
- User Defined Functions
- Machine Learning,
  Advanced Stats and Analytics

@natbusa | linkedin.com: Natalino Busa
Example: Analyze Gowalla events

### Events dataset

<table>
<thead>
<tr>
<th>uid</th>
<th>ts</th>
<th>lat</th>
<th>lon</th>
<th>vid</th>
</tr>
</thead>
<tbody>
<tr>
<td>46073</td>
<td>2010-10-16 12:01:40+0000</td>
<td>51.51339</td>
<td>-0.13101</td>
<td>219097</td>
</tr>
<tr>
<td>46073</td>
<td>2010-10-16 11:20:02+0000</td>
<td>51.51441</td>
<td>-0.141921</td>
<td>23090</td>
</tr>
<tr>
<td>46073</td>
<td>2010-10-13 12:08:19+0000</td>
<td>59.33101</td>
<td>18.06445</td>
<td>136676</td>
</tr>
<tr>
<td>46073</td>
<td>2010-10-12 23:28:43+0000</td>
<td>59.45591</td>
<td>17.81232</td>
<td>96618</td>
</tr>
<tr>
<td>46073</td>
<td>2010-09-15 18:14:13+0000</td>
<td>59.45591</td>
<td>17.81232</td>
<td>96618</td>
</tr>
</tbody>
</table>

### Venues dataset

<table>
<thead>
<tr>
<th>vid</th>
<th>name</th>
<th>lat</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>754108</td>
<td>My Suit NY</td>
<td>40.73474</td>
<td>-73.87434</td>
</tr>
<tr>
<td>249755</td>
<td>UA Court Street Stadium 12</td>
<td>40.72585</td>
<td>-73.99289</td>
</tr>
<tr>
<td>6919688</td>
<td>Sky Asian Bistro</td>
<td>40.67621</td>
<td>-73.98405</td>
</tr>
</tbody>
</table>

### Users dataset

<table>
<thead>
<tr>
<th>uid</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1535</td>
<td>2</td>
</tr>
<tr>
<td>1535</td>
<td>1691</td>
</tr>
<tr>
<td>1535</td>
<td>8869</td>
</tr>
<tr>
<td>1535</td>
<td>23363</td>
</tr>
<tr>
<td>1535</td>
<td>52998</td>
</tr>
<tr>
<td>82694</td>
<td>7860</td>
</tr>
<tr>
<td>82694</td>
<td>8026</td>
</tr>
<tr>
<td>82694</td>
<td>10624</td>
</tr>
<tr>
<td>82694</td>
<td>30683</td>
</tr>
<tr>
<td>82694</td>
<td>57534</td>
</tr>
<tr>
<td>82694</td>
<td>75763</td>
</tr>
<tr>
<td>82694</td>
<td>76154</td>
</tr>
<tr>
<td>82694</td>
<td>76223</td>
</tr>
<tr>
<td>82694</td>
<td>88708</td>
</tr>
<tr>
<td>82694</td>
<td>151476</td>
</tr>
</tbody>
</table>
Spark and Cassandra: distributed goodness

Cassandra: Store all the data
Spark: Analyze all the data

DC1: replication factor 3
Storage!

DC2: replication factor 3

DC3: replication factor 3 + Spark Executors
Analytics!
Cassandra - Spark Connector

Cassandra: Store all the data

Spark: Distributed Data Processing
        Executors and Workers

Cassandra-Spark Connector:

Data locality,
Reduce Shuffling
RDD’s to Cassandra Partitions

DC3: replication factor 3 +
Spark Executors
Joining Tables

val df_venues = cc.sql("select vid, name from lbsn.venues")
               .as("venues")

val df_events = cc.sql("select * from lbsn.events")
               .as("events")

val events = df_events
              .join(df_venues, df_events("events.vid") == df_venues("venues.vid"))
              .select("ts", "uid", "lat", "lon", "venues.vid", "venues.name")

events.count()
Venues and Events
Basic Statistics

```scala
val venues_count = events.groupBy("name").count()
venues_count.sort(venues_count("count").desc).show(5, false)
```

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGA LaGuardia Airport</td>
<td>1679</td>
</tr>
<tr>
<td>JFK John F. Kennedy International</td>
<td>1653</td>
</tr>
<tr>
<td>Times Square</td>
<td>1089</td>
</tr>
<tr>
<td>Grand Central Terminal</td>
<td>1005</td>
</tr>
<tr>
<td>The Museum of Modern Art (MoMA)</td>
<td>392</td>
</tr>
</tbody>
</table>

@natbusa  |  linkedin.com: Natalino Busa
import org.apache.spark.mllib.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

val locs = events.select("lat","lon")
  .map(s => Vectors.dense(s.getDouble(0), s.getDouble(1)))
  .cache()

val numClusters = 20
val numIterations = 20
val clusters = KMeans.train(locs, numClusters, numIterations)
Events k-means clustering
K-Means Scoring

```scala
import org.apache.spark.sql.functions.udf

val func = (lat: Double, lon: Double) =>
    clusters.predict(Vectors.dense(lat, lon))

val sqlfunc = udf(func)

val locs_cid = events
    .withColumn("cluster", sqlfunc(events("lat"), events("lon")))
```
Top venues per cluster

[@natbusa | linkedin.com: Natalino Busa]
Storing k-means clusters top venues in Cassandra

```sql
CREATE TABLE lbsn.topvenues (
    cluster int,
    name text,
    count int,
    PRIMARY KEY (cl,name)
);

locs_cid.select("cluster", "name")
 .groupBy("cluster", "name")
 .agg(Map("name" -> "count"))
 .sort($"cluster", "$COUNT(name)".desc)
 .saveToCassandra("lbsn", "topvenues",
 SomeColumns("cluster", "name", "count"))
```
Storing k-means clusters top venues in Cassandra

```sql
CREATE TABLE lbsn.topvenues (
    cluster int,
    name text,
    count int,
    PRIMARY KEY (cl, name)
);

locs_cid.select("cluster", "name")
    .groupBy("cluster", "name")
    .agg(Map("name" -> "count"))
    .sort($"cluster", $"COUNT(name)".desc)
    .saveToCassandra("lbsn", "topvenues",
                    SomeColumns("cluster", "name", "count"))
```

Works in Spark Streaming too!
Process mining
Process mining
Process mining

<table>
<thead>
<tr>
<th>Source id</th>
<th>Target id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
I SEE AN ADJACENCY LIST
I SEE AN ADJACENCY LIST

IT'S A GRAPH!
Process flow: Sankey Diagrams

- Times Square: 22
- The Museum of Modern Art (MoMA): 54
- Trump Tower: 14
- Grand Central Terminal: 10
- Empire State Building: 8
- Trump Tower: 14
- Times Square: 12
- Empire State Building: 7
- Bryant Park: 7
- Grand Central Terminal: 5
- Other Venues: 9
Flow network: Cord diagram

<table>
<thead>
<tr>
<th>size</th>
<th>id</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4075</td>
<td>Times Square</td>
</tr>
<tr>
<td>1</td>
<td>3173</td>
<td>Empire State Building</td>
</tr>
<tr>
<td>2</td>
<td>3055</td>
<td>The Museum of Modern Art (MoMA)</td>
</tr>
<tr>
<td>3</td>
<td>3041</td>
<td>Columbus Circle</td>
</tr>
<tr>
<td>4</td>
<td>2831</td>
<td>Madison Square Garden</td>
</tr>
<tr>
<td>6</td>
<td>2486</td>
<td>Grand Central Terminal</td>
</tr>
</tbody>
</table>

Circos - [http://circos.ca/](http://circos.ca/)
By Martin Krzywinski
import org.graphframes.GraphFrame

val g = GraphFrame(df_vertices, df_edges)

val results = g.pageRank.resetProbability(0.01).maxIter(20).run()

val results_sorted = results
  .vertices
  .select("id", "pagerank")
  .sort($"pagerank".desc)

results_sorted.head(20)
Venues Importance: Page Rank
Big Data and Fast Data

The diagram illustrates the concept of Big Data and Fast Data, showing how recent data, historical big data, and event streams are distributed across different time periods:

- **Recent Data**: Immediate and recent transactions, customer interactions, etc.
- **Historical Big Data**: Data collected over extended periods, including events, sessions, and customer data.
- **Event Streams**: Continuous data flow over short durations, such as hourly or monthly data streams.

The chart is divided into segments representing time frames of 10 years, 5 years, 1 year, 1 month, 1 day, 1 hour, and 1 month. Each segment highlights how data is categorized based on its time sensitivity and relevance.
Why Fast Data?

1. Relevant up-to-date information.
2. Delivers actionable events.
Why Big Data?

1. Analyze and model
2. Learn, cluster, categorize, organize facts
Batch -> Streaming ETL

Hive -> Flink, Akka, Spark, Kafka
Stream-Centric Architectures

Apache Kafka
A high-throughput distributed messaging system.

@natbusa | linkedin.com: Natalino Busa
Microservices on ML on SQL on Streams

- Ingest Data
- Data Sources, Files, DB extracts
- Batched Data
- Real Time APIs
- Event Streams/Logs
- Streaming Data

- Advanced Analytics
  - read the data
  - write the model

- Distributed Data Store
  - read the model

- Streaming ETL

- Microservices
  - Data Modeling
  - API for mobile and web
  - Alerts and Notifications

- Microservices on ML on SQL on Streams

@natbusa | linkedin.com: Natalino Busa
An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.

Hawkins, 1980

Anomaly detection is the search for items or events which do not conform to the expected pattern

Chandola, 2009
Anomaly Detection

Gaussian Model Based

Parametric Based

Histogram, nonparametric
Clustering with k-means

Intuition:

Distance based, space fully covered
Not a silver bullet though
Clustering users’ venues

**Intuition:**

Users tend to stick in the same places  
People have habits

By clustering the places together  
We can identify anomalous locations

Size of the cluster matters  
More points means less anomalous

Mini-clusters and single anomalies are treated in similar ways ...
Data Science: clustering with DBSCAN

DBSCAN find clusters based on neighbouring density

Does not require the number of cluster k beforehand.
Clusters are not spherical
Data Science: clustering users’ venues

```scala
val locs = checkins_venues.select("uid", "lat","lon")
  .map(s => (s.getLong(0), Seq((s.getDouble(1), s.getDouble(2)) ))
  .reduceByKey(_ ++ _)
  .mapValues( gdbscan cluster _ )
```

Reference: scalanlp/nak
User events DBSCAN clustering
Resources

Spark + Cassandra: Clustering Events

Spark: Machine Learning, SQL frames
https://spark.apache.org/docs/latest/mllib-guide.html

Datastax: Analytics and Spark connector
http://www.slideshare.net/doanduyhai/spark-cassandra-connector-api-best-practices-and-usecases

Anomaly Detection
Datasets

https://snap.stanford.edu/data/loc-gowalla.html
E. Cho, S. A. Myers, J. Leskovec. Friendship and Mobility: Friendship and Mobility: User Movement in Location-Based Social Networks ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2011

The project is being developed in the context of the SInteliGIS project financed by the Portuguese Foundation for Science and Technology (FCT) through project grant PTDC/EIA-EIA/109840/2009.

Pictures:
https://commons.wikimedia.org
Spark ML Tutorial today at 15.15
Capital Lounge 3, second floor